

A Data Broker for Distributed Computing Environments

L. A. Drummond¹, J. Demmel³, C.R. Mechoso², H. Robinson³, K. Sklower³, and J.A. Spahr²

¹ National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA
Ladrummond@lbl.gov

² Department of Atmospheric Sciences, University of California, Los Angeles,
Los Angeles, CA 90095-1565, USA
{mechoso,spahr}@atmos.ucla.edu

³ Computer Science Division, University of California, Berkeley
Berkeley, CA 94720-1776, USA
{demmel,hbr,sklower}@cs.berkeley.edu

Abstract. This paper presents a toolkit for managing distributed communication in multi-application systems that are targeted to run in high performance computing environments; the Distributed Data Broker (DDB). The DDB provides a flexible mechanism for coupling codes with different grid resolutions and data representations. The target applications are coupled systems that deal with large volumes of data exchanges and/or are computationally expensive. These application codes need to run efficiently in massively parallel computer environments generating a need for a distributed coupling to minimize long synchronization points. Furthermore, with the DDB, coupling is realized in a plug-in manner rather than hard-wire inclusion of any programming language statements. The DDB performance in the CRAY T3E-600 and T3E-900 systems is examined

Keywords: MMP systems, Distributed Computing, Data Brokerage, Coupling.

1 Introduction

The Distributed Data Broker (DDB) is a toolkit for managing distributed communication in multi-application systems that run coupled in high performance computing environments. The DDB evolved from a Data Broker designed as a part of a coupled atmosphere-ocean modeling system, in which the model components can work on different horizontal resolutions, grid representations and cover different geographical domains [1]. The high efficiency demanded by those codes in massively parallel computer environments generated a need for extending the Data Broker in a way that minimizes long synchronization points inside model components and memory bottlenecks. Using the DDB, applications are integrated to the coupled

system in a plug-in manner rather than by hard-wire inclusion of any programming language statements. The DDB was designed under a consumer-producer paradigm, in which, an application produces data to be consumed by one or more applications, and an application can be a consumer, producer or both.

This paper is an introduction to the DDB tool. Section 2 presents a summary of the functionality of the DDB and its library components. A general example of a coupled application using the tool is described in Section 3. Performance results are shown in Section 4.

2 The Distributed Data Broker.

The functionality of the DDB is encapsulated in a modular design that contains three libraries of routines that are built to work together and called on demand from different places inside the codes to be coupled. These DDB components are the Communication Library (CL), The Model Communication Library (MCL) and the Data Translation Library (DTL). The CL is the core library of routines that it is used to implement the point-to-point communication between computational nodes in a distributed environment. This DDB component encapsulates the functionality of widely used message passing software like PVM 3 or MPI into the DDB context. A more technical description of the CL is presented in [3]

Two types of steps characterize the coupling of applications using the DDB; an initial registration step and subsequent data communication steps. The MCL provides an API (Application Programming Interface) that supports the implementation of both steps from C or Fortran programs. The registration step is an initial code “handshake” in which different codes exchange information about the production and consumption of data. The registration step begins with the identification of a process or a task as the Registration Broker (RB) with a call to the `MCLiamRegisgtrationBroker` routine. There is only one RB per coupled run and this task only exists during the registration step. This implies that after the registration step, the process acting as the RB can perform any other tasks inside one of the applications being coupled. In addition, each application must identified a control process (CP), each CP is responsible for reporting global domain information to the application like grid resolution, number of processes, data layout, and frequency of production or consumption to the RB. The RB can also be the CP for the application that spawns it.

The RB starts collecting information from all the CP's. Then, the RB processes this information to match producers against consumers. For example if Process 1 of Model A is designated as the RB as in Fig. 1, this collects general information from process 1 of Model B (such as global domain, grid resolution, number of processes, and frequencies of production and consumption. Without loss of generality, in this example we depicted one model that works in a wider domain than the other, and uses a different horizontal grid spacing. The DDB will also works with equal domains, and equal grid spacing as long as a geographical domain can be mapped into the other domain and a grid translation function exist between both grids.

Lastly, the registration step ends with a call to `MCLRegistration` from all other processes participating in the coupling to register their process id and subdomain.

Every process receives back a list of processes that it will exchange data with at execution time. As a result, every participating process in the coupling has enough information to send and receive data from its peers without the need of a centralized entity regulating the exchanges of information.

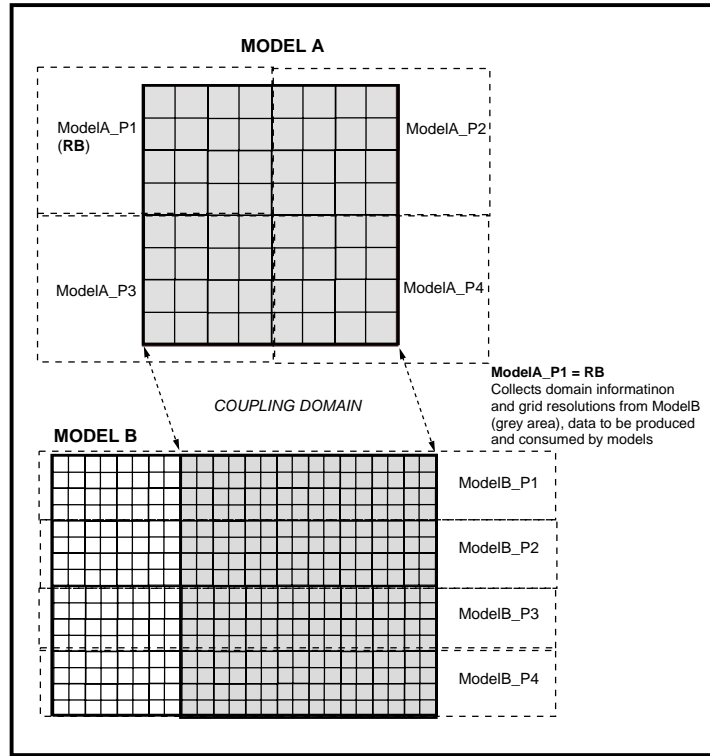


Fig. 1. DDB registration step. Registration Broker (*RB*) collects information of every model (i.e., *model A* and *model B*). This information includes: model's resolution, domain, offers for data production, requests for data production, frequency of consumption and production of data, and parallel data layout.

Fig. 2 presents an schematic of all the MCL routines that implement the registration and communication steps. The communication step is characterized by patterns of communication between the coupled components. A producer code that wants to send data to its consumers, will simply execute a call to `MCLSendData`, which gets translated into several CL commands that in turn call the MPI or PVM libraries to complete the communication step. Thus, the MCL-CL interface provides a level of transparency and code portability because the communication syntax used inside a program remains invariant when porting the code from PVM 3 to MPI or vice-versa and these communication packages in turn provide portability across platforms.

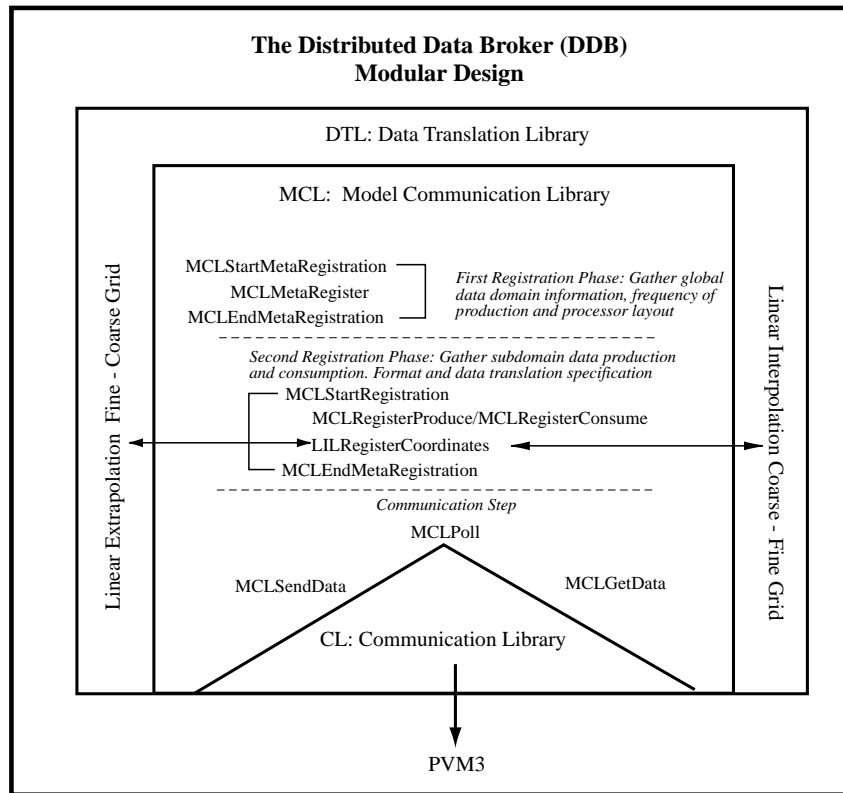


Fig. 2. Schematic of the DDB. The Application Programming Interface is provided via the MCL. In turn the MCL makes use of the CL library to interface with standard message passing libraries like PVM and the user-defined Data Translation Libraries. The current DDB has implemented a Linear Interpolation Library (LIL) of routines

The basic MCL communication phase has two operations, `MCLGetData` and `MCLSendData`. A user's call to `MCLSendData` automatically generates one or many calls to the send-routine in the CL library, one per consumer of the data produced (e.g., one `pvmfsend` per consumer). Similarly, a user's call to `MCLGetData` automatically receives one or many messages, pastes them together and transforms them into compatible data for the consumer's grid using a predefined DTL routine. The DTL component handles the data transformations from a producer's grid to the consumer's grid. The DTL routines are invoked by certain calls to the MCL that deliver data at the consumer end (i.e., `MCLGetData`). The DTL can include several numerical transformation routines and the user can decide the transformation algorithm to be used according to the numerical requirements of the applications. In any case, the calls to the MCL library remain the same but each of

the low-level transformation routines in the DTL are overloaded with different procedures depending on the context. In view of our current coupling scenarios and requirements for data transformations, we have implemented a set of linear interpolation routines.

3. An Example of Coupling with the DDB.

The current version of the Distributed Data Broker (DDB) is being used to couple different model components of the UCLA Earth System Model (ESM) under the NASA/ESS HPPC program. In this system the model components are parallel codes that in turn run in parallel exchanging atmospheric or oceanic fields in a prescribed time intervals. In conventional couplers, these data exchanges and translations are handled using a centralized global domain algorithm. Here we present a fully distributed approach to coupling in which the data translations between models are handled in parallel and using a subdomain based numerical algorithms. The DDB approach to coupling promotes high levels computational efficiency by reducing the number of synchronization points, the need of global reductions operations, and idle nodes in the system.

The UCLA Atmospheric General Circulation Model (AGCM) is a state of the art grid point model of the global atmosphere ([2],[5]) extending from the Earth's surface to a height of 50 km. The model predicts the horizontal wind, potential temperature, water vapor mixing ratio, planetary boundary layer (PBL) depth and the surface pressure, as well as the surface temperature and snow depth over land. The Oceanic General Circulation Model is the Parallel Ocean Program (POP), which is also based on a two-dimensional (longitude-latitude) domain decomposition [4], and uses message passing to handle data exchanges between distributed processors.

The UCLA AGCM is a complex code representing many physical processes. Despite the complexity of the code, one can identify the following two major components:

- **AGCM/Dynamics**, which computes the evolution of the fluid flow governed by the appropriate equations (the primitive equations) written in finite differences.
- **AGCM/Physics**, which computes the effect of processes not resolved by the model's grid (such as convection on cloud scales) on processes that are resolved by the grid (such as the flow on the large scale).

The OGCM also has two major components:

- **OGCM/Baroclinic**, determines the deviation from the vertically averaged velocity, temperature and salinity fields.
- **OGCM/Barotropic**, determines the vertically averaged distributions of those fields.

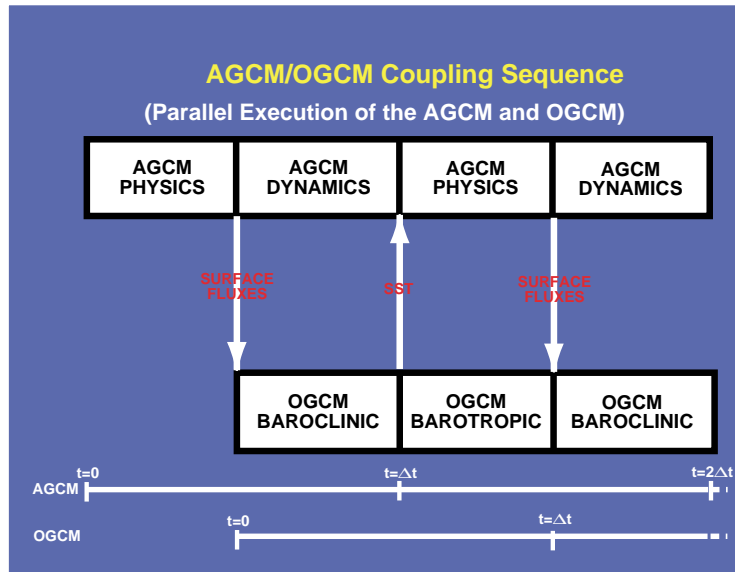


Fig. 3. Distributed AGCM-OGCM coupling. The AGCM send surface fluxes to the OGCM and receives in return Sea Surface Temperature. This exchanges happen at regular intervals Δt

The coupled atmosphere-ocean GCM, therefore, can be decomposed into four components. When run on a single node the AGCM and OGCM codes execute sequentially and exchange information corresponding to the air-sea interface. The AGCM is first integrated for a fixed period of time and then transfers the time-averaged surface wind stress, heat and water fluxes to the OGCM. This component is then integrated for the same period of time and transfers the sea surface temperature to the AGCM. The data transfers, including the interpolations required by differences in grid resolution between model components, was originally performed by a suite of coupling routines and we refer to this approach as the centralize coupling approach. Coupling with the DDB is realized with a registration step followed by model computations and inter-model communication handled by MCLGetData and MCLSendData calls. The necessary data translations are also performed under these calls.

The coupled GCM runs in a parallel environment following the scheme depicted in Fig. 3, which allows the two codes to run in parallel. Because there are no data dependencies between the AGCM/Dynamics and the OGCM/Baroclinic, these components can run in parallel. Further, AGCM/Physics can start as soon as OGCM/Baroclinic completes its calculation, because this module provides the sea surface temperature. Similarly, The AGCM/Physics can run in parallel with OGCM/Barotropic.

4. Performance Results.

This section presents some results obtained from running the coupled UCLA AGCM/OGCM model described in section 3. We compare here the centralized coupling against the decentralized one. Fig. 4 to Fig. 6 shows the model resolutions used in each case, and compare the memory and time required by the coupling interfaces

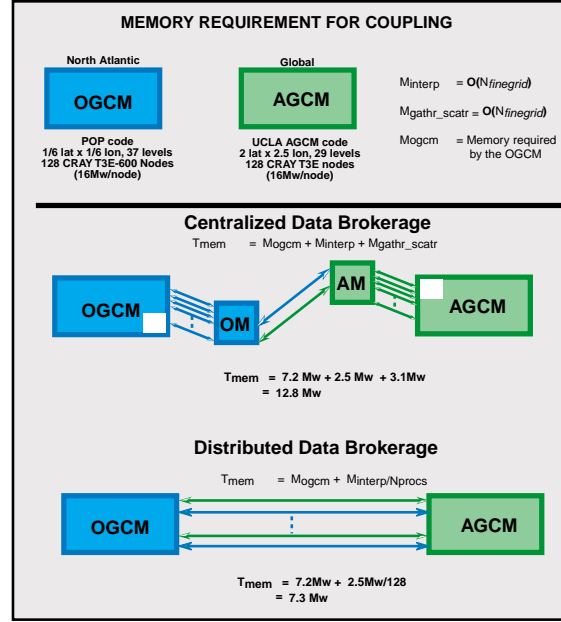


Fig. 4. Memory requirements for centralized and distributed coupling.

Fig. 4 and Fig. 5 illustrate comparison results based on the memory requirements for both coupling implementations. In Fig. 4, the centralized data brokerage requires almost twice as much memory as the distributed data brokerage because it needs to collect the entire grid from one model in a single node. In the distributed case, each processor has enough information to produce the data needed by consumer processes and communication is realized in distributed manner. In Fig. 5, a more drastic scenario is presented, in which the centralized coupling cannot be realized because of the 45Mw memory requested in a single computational node. In this case the distributed case requires less than a third of the memory requested by the centralized approach.

Fig. 6 compares the execution time between the two coupling approaches, and in this case the AGCM is sending 4 fields to the OGCM, and the requested time by the distributed approach is one third of the centralized. In the reverse communication, the

OGCM sends a single field to the AGCM and the requested time is also greatly reduced with the distributed approach.

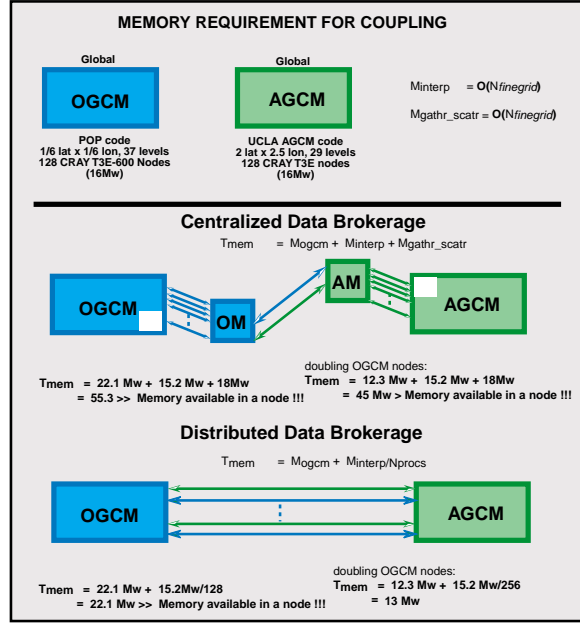


Fig. 5. Memory requirements for centralized and distributed coupling. First, we double the resolution OGCM resolution and increase the number of nodes

Fig. 7 presents the asymptotic behavior of centralized vs. distributed coupling. As indicated the number of seconds required by coupling the AGCM/OGCM in the centralized case (one process case) grows exponentially as the problem size is increased. The time requested by the distributed coupling approach, the DDB, is reduced as the number of processes is also increased.

5. Conclusions.

As computational sciences continue to push forward the frontier of knowledge about physical phenomena, more complex models are and will be developed to enable their computerized simulations. The demand for computational resources to carry out these simulations will also increase, as well as the need for optimized tools that help application developers to make better use of the available resources. The DDB addresses not only the issues of optimal coupling, but also provides a flexible approach to coupling models and applications in a “plug-and-play” manner rather than intrusive coding in the applications.

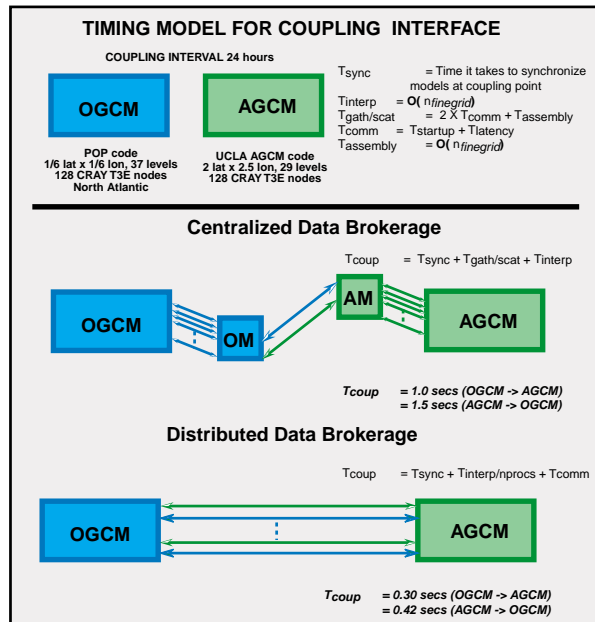


Fig. 6. Simplified Timing model of centralized vs. distributed coupling.

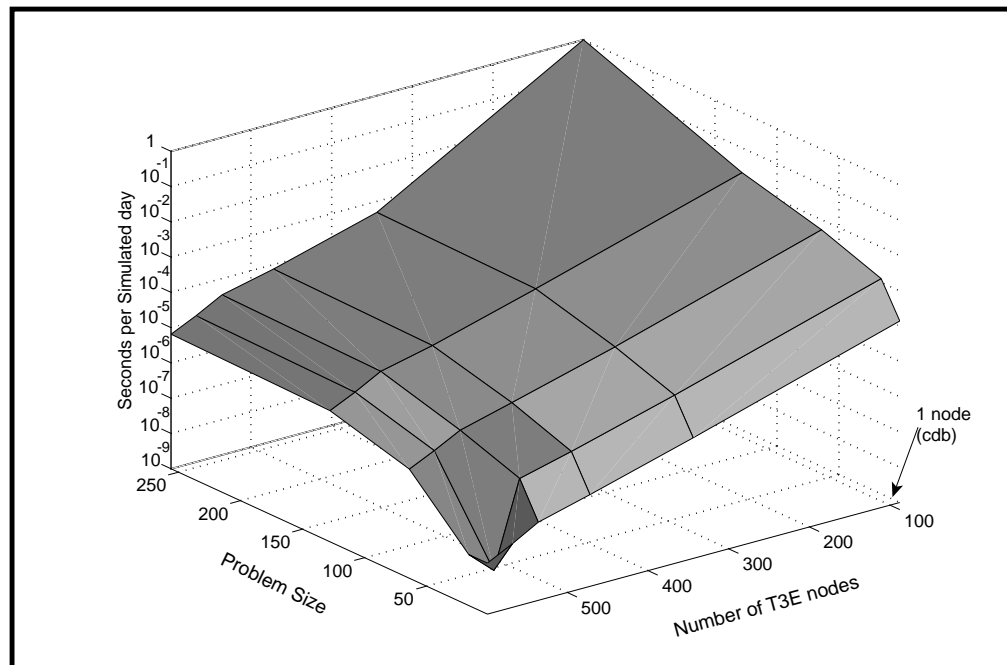


Fig. 7. Asymptotic behavior of centralized vs. distributed coupling .

Further development of the DDB is still under way at University of California, Los Angeles and collaborators at UC Berkeley. Future agenda includes the inclusion of higher order interpolations for data translations, use of other communication libraries such as MPI, and continue to prototype other scientific applications using the DDB technology.

Acknowledgements.

This project has been supported by the NASA High Performance Computing and Communication for Earth and Space Sciences (HPCC-ESS) project under CAN 21425/041. The tests were performed at the Department of Energy's National Energy Research Scientific Computing center (NERSC)

REFERENCES.

1. Drummond, L. A., J. D. Farrara, C. R. Mechoso, J. A. Spahr, J. W. Demmel, K. Sklower and H. Robinson, 1999: An Earth System Model for MPP environments: Issues in coupling components with different complexities. *Proceedings of the 1999 High Performance Computing - Grand Challenges in Computer Simulation Conference* April 11-15, 1999, San Diego, CA, 123-127.
2. Mechoso, C. R., L. A. Drummond, J. D. Farrara, J. A. Spahr, 1998: The UCLA AGCM in high performance computing environments. *In Proceedings, Supercomputing 98*, Orlando, FL.
3. Sklower, K., H.R. Robinson, L.A. Drummond, C.R. Mechoso, J. A. Spahr, E. Mesrobian, 2000: The Data Broker: A decentralized mechanism for periodic exchange of fields between multiple ensembles of parallel computations
<http://www.cs.berkeley.edu/~sklower/DDB/paper.html>
4. Smith, R.D., J.K. Dukowicz, and R.C. Malone, 1992: Parallel Ocean General Circulation Modeling, *Physica D*, 60, 38-61.
5. Wehner, M. F., A. A. Mirin, P. G. Eltgroth, W. P. Dannevik, C. R. Mechoso, J. D. Farrara and J. A. Spahr, 1995: Performance of a distributed memory finite-difference atmospheric general circulation model. *Parallel Computing*, 21, 1655-1675.